Statistical Learning of Multi-Dimensional Textures

A thesis submitted in fulfillment
of the requirements for the degree of
Master of Science

by

Ziv Bar-Joseph

supervised by
Dr. Dani Lischinski

Institute of Computer Science
The Hebrew University of Jurusalem
Jurusalem, Israel

June 9, 1999

Acknowledgments

I would like to thank my supervisor Dr. Dani Lischinski for suggesting this interesting problem, providing the initial direction to the solution and helping me throughout this work.

I thank Prof. Michael Werman for his insights regarding computer vision and Dr. Ran El-Yaniv for introducing me to statistical learning.

All the three mentioned above contributed greatly to this work during our weekly Thursday meetings. Much of the work described in this thesis was done during those meetings.

I want to thank my friends in the computer graphics lab. Especially I thank Iddo for his enthusiasm from this work, it almost caught me too.

Finally, I want to thank my family for supporting me. I learned about the fascination of research from home, and I thank my parents for showing me that.

Contents

List of Figures

Abstract

We present an algorithm based on statistical learning for synthesizing static and time-varying textures matching the appearance of an input texture. Our algorithm is general and automatic, and it works well on various types of textures including 1D sound textures, 2D texture images and 3D texture movies. The same method is also used to generate 2D texture mixtures that simultaneously capture the appearance of a number of different input textures. In our approach, an input textures is treated as a sample signal generated by a stochastic process. We first construct a tree representing a hierarchical multi-scale transform of the signal using wavelets. From this tree, new random trees are generated by learning and sampling the conditional probabilities of the paths in the original tree. Transformation of these random trees back to signals results in new textures. In the case of 2D texture synthesis our algorithm produces results that are generally as good as those produced by earlier works in this field. For texture mixtures our results are better and more general than those produced by earlier works. For texture movies, we present the first algorithm that is able to automatically generate movie clips of dynamic phenomena such as waterfalls, fire flames, a school of fish, a crowd of people, etc. A one-dimensional variant of our algorithm is able to synthesize various sound textures, such as traffic, water sounds, etc. Our results indicate that the proposed technique is effective and robust.

1 Introduction

Texture synthesis is an interesting and important problem in the field of computer graphics. Recently, several techniques have emerged in the computer graphics literature that are able to analyze an input texture sample and synthesize many new random similar-looking textures [9, 19, 38]. This work extends these techniques in several important ways. First, we describe a new *texture mixing* algorithm — a statistical learning algorithm that operates on *several different* input texture samples to synthesize a new texture. This new texture is statistically similar to all of the input samples and it exhibits a mixture of their features. Second, we extend our approach from the domain of static textures to the domain of *texture movies*: dynamic, time-varying textures, or TVTs for short. More specifically, we present a technique capable of generating a sequence of frames, corresponding to a temporal evolution of a natural texture or pattern, that appears similar to an input frame sequence. For example, using this technique we have been able to generate short movies of various dynamic phenomena, such as waterfalls, fire flames, a school of jellyfish, turbulent clouds, an erupting volcano, and a crowd of people. The generated sequences are distinguishable from the original input sample, yet they manage to capture the essential perceptual characteristics and the global temporal behavior observed in the input sequence. A specialized version of this method, described in a separate paper [2], is able to synthesize 1D *sound textures*, such as sounds of traffic, water, etc.

The natural applications of texture movie synthesis are in the areas of special effects for motion pictures and television, computer-generated animation, computer games, and computer art. Our method allows its user to produce many different movie clips from the same input example. Thus, a special effects technical director should be able to fill an entire stadium with ecstatic fans from a movie of a small group, or populate an underwater shot with schools of fish. Designers of 3D virtual worlds should be able to insert animations of clouds, smoke, and water from a small number of input samples, without ever repeating the same animation in different places. However, these are by no means the only applications of such a technique. For example, methods for statistical learning of 2D texture images have been successfully applied not only to texture synthesis, but also to texture recognition and image denoising [11]. These applications are made possible by realizing that statistical learning of 2D textures implicitly constructs a statistical model describing images of a particular class. Similarly, our approach for learning TVTs can be used as a statistical model suitable for describing TVTs. Therefore, it should be possible to apply this statistical model for tasks such as classification and recognition of such movie segments.

1.1 Overview of Approach

Texture images are examples of approximately stationary 2D signals. We assume such signals to be generated by stochastic processes. This thesis introduces a new approach for statistical learning of such signals:

1. Obtain one or more training samples of the input signals.
2. Construct a hierarchical multi-resolution analysis (MRA) of each signal sample. Each MRA is represented as a tree, assumed to have emerged from an unknown stochastic source.
3. Generate a new random MRA tree by statistically merging the MRA trees of the input samples.
4. Transform the newly generated MRA back into a signal, yielding a new texture that is statistically and perceptually similar to each of the inputs, but at the same time different from them.

Note that the procedure outlined above is applicable to general n-dimensional signals, although it is practical only for small values of n (since its time and space complexities are exponential in n).

Since the tree merging algorithm in stage 3 above involves random choices, each invocation of the algorithm results in a different output texture. Thus, many different textures can be produced from the same input. If all of the input samples are taken from the same texture, we obtain a 2D texture synthesis algorithm similar to that of De Bonet [9]. If the input samples come from different textures, the result is a mixed texture.

A naive extension of the above approach to generation of TVTs would be to independently synthesize a new frame from each frame in the input sequence. However, this method fails to capture the temporal continuity and features of the input segment. In contrast, the approach presented in this thesis is to synthesize all three dimensions of the TVT simultaneously (although the temporal dimension is processed differently from the spatial dimensions).

As in the 2D texture case, we assume that a time-varying texture is generated by a stochastic process. It is a 3D signal $S(x, y, t)$, where x and y are the spatial coordinates of the signal (pixel coordinates in each frame), and t is the temporal coordinate (frame number). By applying the approach outlined above to 3D signals, with an appropriately chosen MRA scheme, we obtain a statistical learning algorithm for TVTs.

1.2 Thesis Outline

The rest of this thesis is organized as follows. In the next section we provide definition that will allow the reader to better understand the terms we will be using, and will provide a common language that will be used in this thesis. In Section 3 we review the previous work and the mathematical background needed for describing our algorithm. Section 4 presents the statistical learning algorithm. In Section 5 we show the implementation of our algorithm to the task of synthesizing 2D textures, we also show how to use our algorithm to produce “texture mixing”. In Section 6 we show the implementation of our algorithm to the synthesis of sound textures and texture movies. Section 7 concludes this thesis and discusses our results and directions for future work.

2 Definitions

Before we continue to describe our work, we first define some terms that will be used in the following sections.

2.1 Textures

There exists no precise definition of texture. In this section we will try to present some of the definitions that appear in the literature, and show how these definitions relate to the definition of textures that we meant. We will also extend this definitions (that were given to describe 2D textures), and apply them to sound textures and texture movies.

A good source for definitions of textures can be found on the web in [1]. Here we bring some of the definitions that appear in that site. Jain in [20] defines textures as:
The term texture generally refers to repetition of basic texture elements called texels. The texel contains several pixels, whose placement could be periodic, quasi-periodic or random. Natural textures are generally random, whereas artificial textures are often deterministic or periodic. Texture may be coarse, fine, smooth, granulated, rippled, regular, irregular, or linear.

Another definition can be found in a book by Wilson and Spann [34].
Textured regions are spatially extended patterns based on the more or less accurate repetition of some unit cell (texton or subpattern).

The definition that best matches our view of textures comes from Smith in [29]. He gives the following definition.
Textures are homogeneous patterns or spatial arrangements of pixels that regional intensity or color alone does not sufficiently describe. As such, textures have statistical properties, structural properties, or both. They may consist of the structured and/or random placement of elements, but also may be without fundamental subunits.

We extend this definition in the following way. As will be described in Section 3.2, we use statistical learning algorithm in order to synthesize new textures. In order to use such an algorithm on a given sample, we claim the following:
A Texture is a signal that exhibit the following property. Using any window of size larger than some critical size, the "information content" exhibited in the window is invariant to the window's position within the given sample.
See Figure 1 for examples of 2D textures.

2.2 TVT

The last definition allows us to use the term texture not only for describing a particular vatiety of 2D images, but also for sound textures and texture movies. These signals share a common property: they are both *time-varying textures*, or TVTs for short. In these signals the claim that all the information is contained in a small sample is also true. For example, a drum beat repeats itself several times within a short sound segment. The same thing is true for a waterfall video sequence. In a waterfall sequence the motion of the falling water repeats itself. Another thing in common between these signals and 2D textures is the fact that they too usually consist of fundamental subunits (such as a car horn in a traffic jam sound or a single cloud in a sky full of clouds). For this reasons we thought that they would be appropriate for statistical learning, and indeed this was the case. Example of a waveform of a sound texture is shown in Figure 2.

In the next two sections, when we refer to textures, we mean all kinds of textures (2D as well as TVTs), unless we specifically state differently. The implementation of our algorithm to synthesizing TVTs is explained in detail in Section 6.

2.3 MRA

MRA stands for *multi-resolution analysis*. This analysis on an input texture is usually done using wavelets as described in Section 3.3. The result of such an analysis is a tree that represents the input texture. Our algorithm for statistical learning operates on the MRA representation of the input texture. Thus, in order to implement our algorithm we first transform the input texture into its MRA (tree) representation, and apply the algorithm on this representation. The output of the algorithm is also an MRA representation of the output texture, and an inverse transform is applied in order to generate the output texture. Our algorithm is independent of the specific representation chosen. It is general, and can be applied to any MRA the user wishes to work with. However, choosing the right representation is a key issue when trying to produce good output textures. We worked with different MRAs in order to produce the three types of textures we discuss in this work. For the sound textures we use an MRA which is a binary tree as described in Section 6. For the 2D textures we use a quad tree as described in Section 5. The texture movies are represented using a combination of octrees and quad trees as will be discussed in Section 6. Another difference between the MRAs used for the three types of textures we describe in this work are the filters we used. The specific filters we used for each type are described in detail in Section 3.3.

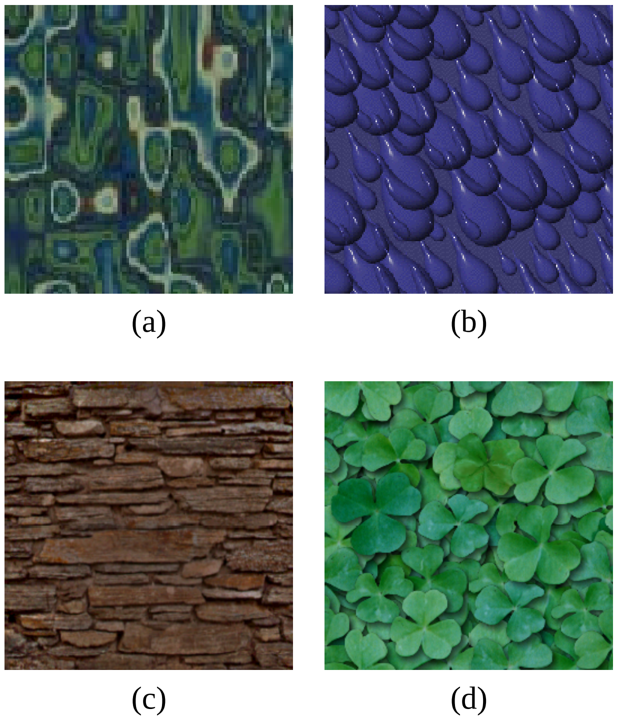


Figure 1: 2D texture example. (a) and (b): synthetic textures of surface and drops. (c) and (d): natural textures of bricks and plants. Note that all this textures consist of fundamental subunits.

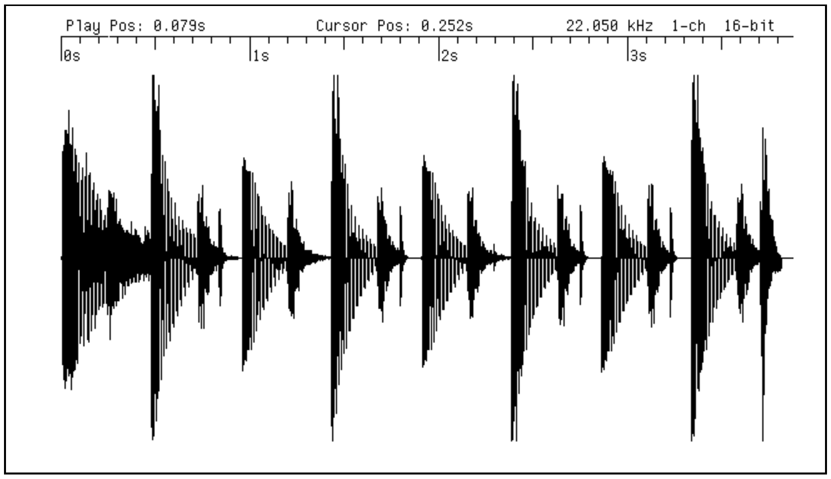


Figure 2: Sound texture example in which the waveform of a drum beat is shown. Again, it can be seen that this sound segment consist of repeating subunits.

3 Previous Work and Background

In this section we review the previous work that has been done on texture synthesizing. We also supply the mathematical background that is needed in order to explain our statistical learning algorithm and the MRA construction.

3.1 Previous Work

Most of the previous work on 2D texture synthesis has focused on the development of procedural textures[15, 24, 33, 35, 36].

More recently work has been done which considers textures as sample from probabilistic distribution. The first methods were suggested by Zhu *et al* [38] and Heeger and Bergen [19]. Both these works iteratively resample random noise to coerce it into having particular multiresolution histograms that matches the histograms of the input sample. De Bonet in [10] analyzes their approach and shows several cases in which the output fails to capture the appearance of the input texture. The major drawback of their approach is the fact that even if two images have the same subband histograms, they could have very different appearance because their joint occurrences does not match.

To solve this problem De Bonet in [10, 9, 11] suggests a method that synthesizes 2D textures using a multiresolution sampling procedure. His work is based on the fact that images perceived as textures contain regions which differ by less than some discrimination threshold, and randomization of these regions does not change the perceived characteristics of the texture. His method works in a two-phase process. The input texture is first analyzed by measuring the joint occurrence of texture discrimination features at multiple resolutions. In the second phase, a new texture is synthesized by sampling successive spatial frequency bands from the input texture, conditioned on the similar joint occurrence of features at lower spatial frequencies. Our work can be viewed as extensions of De Bonet's approach [9] to multiple input samples and to time-varying textures.

Texture mixing, a process of generating a new texture that contains features from several different input textures, has been addressed in several previous works. Burt and Adelson [6] produce smooth transitions between different textures by weighted averaging of the Laplacian pyramid coefficients of the textures. This technique is very effective for seamless image mosaicing, but is less suitable for producing a mix of textures across the entire image, as will be demonstrated in section 5.2. Heeger and Bergen [19] use their histogram-based texture synthesis algorithm to generate texture mixtures in which the color comes from one texture, while the frequency content comes from another. In this work we produce a different kind of mixtures, in which both colors and frequencies are mixed together.

To our knowledge, there have not been any previous attempts towards statistical learning of TVTs from input samples. So far, synthesis of dynamic natural phenomena has mostly been possible only via computationally intensive physically based simulations. For example, steam, fog, smoke, and fire have been simulated in this manner [16, 14, 30, 31]. Explosions, fire, and waterfalls have been successfully simulated by animated particle systems [25, 26, 28]. Simplified physically-based models have also been used to produce synthetic waves and surf [18, 23]. While the techniques mentioned above have been able to generate impressive results of compelling realism, a custom-tailored model must be developed for each type of simulated phenomena. Furthermore, such simulations are for the most part expensive and, more importantly, the resulting animations can be difficult to control. In contrast, as we shall see, statistical learning of TVTs is extremely general, automatic, and fast alternative, provided that an example of the desired result is available.

Heeger and Bergman [19] have applied their texture synthesis technique to the generation of 3D solid textures (in addition to 2D texture images). However, in a solid texture all three dimensions are treated in the same manner, whereas in a 3D TVT the temporal dimension must be treated differently from the two spatial dimensions, as shall be explained in Section 6. Another difference between their work and ours is that their method learns the statistical properties of a 2D texture and the generates a 3D texture with the same properties, whereas our method analyzes a 3D signal (TVT) directly.

Sound synthesis is a wide and fertile research area. Since our primary interest in this work is in the field of computer graphics, we shall not attempt to review the sound synthesis literature here. The interested reader in refered to [2] for a detailed description of the sound synthesis algorithm.

3.2 Statistical Learning

We first informally define and explain a few terms needed for our discussion. One of the main tasks in *statistical learning* is the estimation of an unknown stochastic source given *samples*, which are examples from the source. For instance, a sample can be a complete movie, a 2 dimensional texture, a sound segment, etc. A good statistical model not only fits the given examples but generalizes to generate previously unseen examples. Generalization is a key issue in learning. A model with good generalization can generate new random examples with a probability distribution that resembles that of the unknown source. *Sampling the model* means that we generate such new random examples from a learned model.

3.2.1 The Mutual Source

Our basic assumption is that multi-dimensional signals such as texture images and texture movies, are random samples of an unknown stochastic source. Our goal is to learn a statistical model of this source given a small set of training examples.

Consider signal examples $s_1, s_2, \ldots, s_k$ where each of the s_i is assumed to be an example from a stochastic source S_i. Although the sources S_i are unknown we assume that s_i is a typical example from that source conveying its essential statistics. Our task is to estimate the statistics of a hypothetical source $Z = Z(S_1, \ldots, S_k)$ called the *mutual source* of $S_1, \ldots, S_k$. Intuitively, the source Z is the "closest" (in a statistical sense) to all the S_i simultaneously.
More formally, we define the mutual source in the following way. Let P and Q be two distributions. Their mutual source Z is defined as the distribution that minimizes the Kullbak-Leibler (KL) divergence [7] to both P and Q. The KL-divergence from a distribution Z to a distribution P, where both P and Z are defined over a support A, is defined to be $D_{KL}(P||Z) = \sum_{a \in A} P(a) \log \frac{P(a)}{Z(a)}$. Specifically,

$$Z = \arg\min_{Z'} \lambda D_{KL}(P||Z') + (1 - \lambda) D_{KL}(Q||Z').$$

The parameter λ should reflect the prior importance of P relative to Q. (When no such prior exists one can take $\lambda = 1/2$). The expression $\min_{Z'} \lambda D_{KL}(P||Z') + (1 - \lambda) D_{KL}(Q||Z')$ is known as the Jensen-Shannon dissimilarity. Using convexity arguments it can be shown (see e.g. [17]) that the mutual source is unique, and therefore, the Jensen-Shannon measure is well defined.

After learning the mutual source Z, we can sample from it and synthesize from it "mixed" signals that are statistically similar to each of the sources S_i (and the examples s_i). In the special case where the samples $s_1, \ldots, s_k$ originate from the same source S (that is $S = S_1 = \cdots = S_k$), the mutual source Z is exactly S and when we sample from Z we synthesize new random examples that resemble each of the given examples s_i.

3.2.2 Choosing the Representation

A key issue when modeling, analyzing and learning signals is the choice of their representation. For example, a signal can be represented directly by its values. Alternatively, it can be represented in the frequency domain via the Fourier transform. These two representations are the most common for modeling 1D and 2D signals and there are various, well established approaches to estimating the underlying unknown source with respect to such representations (see e.g. [5, 22]). Although the use of such representations has been successful in many application domains, they are not adequate for representation and analysis of "textural signals" such as the ones treated in this work. Such signals typically contain detail at different scales and locations. Many recent research efforts suggest that a better representation for such signals are multi-resolution structures, such as wavelet-based representations (see e.g. [3, 37]). The papers [4, 3] provide a theoretical formulation of statistical modeling of signals by multi-resolution structures. In particular, these results consider a generative stochastic source (that can randomly generate multi-resolution structures), and study their statistical properties. They show that stationary signals (informally, a signal is stationary if the statistics it exhibits in a region is invariant to the region's location) can be represented and generated in a hierarchical order, where first the coarse, low resolution details are generated, and then the finer details are generated with probability that only depends on the already given lower resolution details. In particular, hierarchical representation was already successfully applied to modeling the statistics of two-dimensional textures as stated above [9, 11].

We also adopt this view and develop an algorithm that learns the conditional distributions of the mutual source. We transform the given examples to their multi-resolution, tree representations and learn the conditional probabilities along paths of the mutual source tree, using an estimation method for linear sequences. Thus, we are left with the problem of learning mutual sources of sequences, which are simply paths in the representing tree.

3.2.3 Statistical Learning of Sequences

The particular estimation algorithm for sequences we chose to use is an extension of the algorithm from [17] which operates on sequences over a finite alphabet, as opposed to real numbers. Given a sample sequence S, this algorithm [17] generates new random sequences which could have been generated from the source of S. In other words, based only on the evidence of S, each new random sequence is statistically similar to S. The algorithm generates the new sequence without explicitly constructing a statistical model for the source of S. This is done as follows: suppose we have generated s^i, the first i symbols of the new sequence. In order to choose the next, $(i+1)$-st symbol, the algorithm searches for the longest suffix of s^i in S. Among all the occurrences of this suffix in S, the algorithm chooses one such occurrence x uniformly at random and chooses the next symbol of s^i to be the symbol appearing immediately after x in S. This algorithm has been adapted to work on paths of tree representations of the signals treated in this work, as described in Section 4.

3.3 Wavelets and Steerable Pyramids

Wavelets have become the tool of choice in analyzing single and multi-dimensional signals, especially if the signal has information both at different scales and localizations. The fundamental idea behind wavelets is to analyze the signal's local frequency at all scales and locations, using a fast invertible hierarchical transform. Wavelets have been effectively utilized in many fields, including video multimedia databases and coders, radar, magnetic resonance and ultrasound imaging, stochastic resonance, solitons for local area communications, high resolution image synthesis, financial data analysis, industrial measurements, and many more. For a comprehensive review of wavelets applied to computer graphics see [32].

The wavelet is a multi-scale decomposition of the signal and can be viewed as a complete tree, where each level stores the projections of the signal, with the wavelet basis functions of a certain resolution (at all possible translations of the basis functions).

The wavelets we use in this work to analyze the input textures and generate the different MRAs are produced by using the Daubechies wavelets [8] and the *steerable pyramids* [27]. Daubechies wavelets are used in the time domain and steerable pyramids are used in the image domain. Thus, for the sound textures we use only the Daubechies wavelets, and for the 2D textures we use only the steerable pyramids (as described in Section 5). For the texture movies we use a combination of both as described in Section 6.

The Daubechies wavelet is a one dimensional wavelet that produces a series of coefficients that describe the behavior of the signal at dyadic scales and locations. The Daubechies wavelet is produced as follows: Initially, the signal is split into lowpass/scaling coefficients by convolving the original signal with a lowpass/scaling filter (denoted in this work by Φ) and the wavelet/detail coefficients are computed by convolving the signal using a Daubechies wavelet filter (denoted Ψ). Both responses are subsampled by a factor of 2, and the same filters are applied again on the scaling coefficients, and so forth.

The steerable pyramid is a multi-scale, multi-orientation linear signal decomposition. This wavelet has many superior properties compared to traditional orthonormal wavelets, especially with respect to translation and rotation invariance, aliasing and robustness due to its nonorthogonality and redundancy. The steerable filter is produced as follows: Initially, the signal is split into low and highpass subbands. The lowpass subband is then split into a set of k oriented bandpass subbands using k oriented filters (denoted in this work by Ω_i) and a new lowpass subband is computed by convolving with a lowpass filter (denoted Θ). This lowpass subband is subsampled by a factor of 2 in each direction and the resulting subsampled signal is processed recursively (see Figure 3). A detailed exposition on steerable pyramids and further references can be found in [27].

The wavelets can be transformed back into the original signal using a fast hierarchical transform. The computation proceeds from the root of the tree down to the leafs, using filters that are complementary to those used in the wavelet transform.

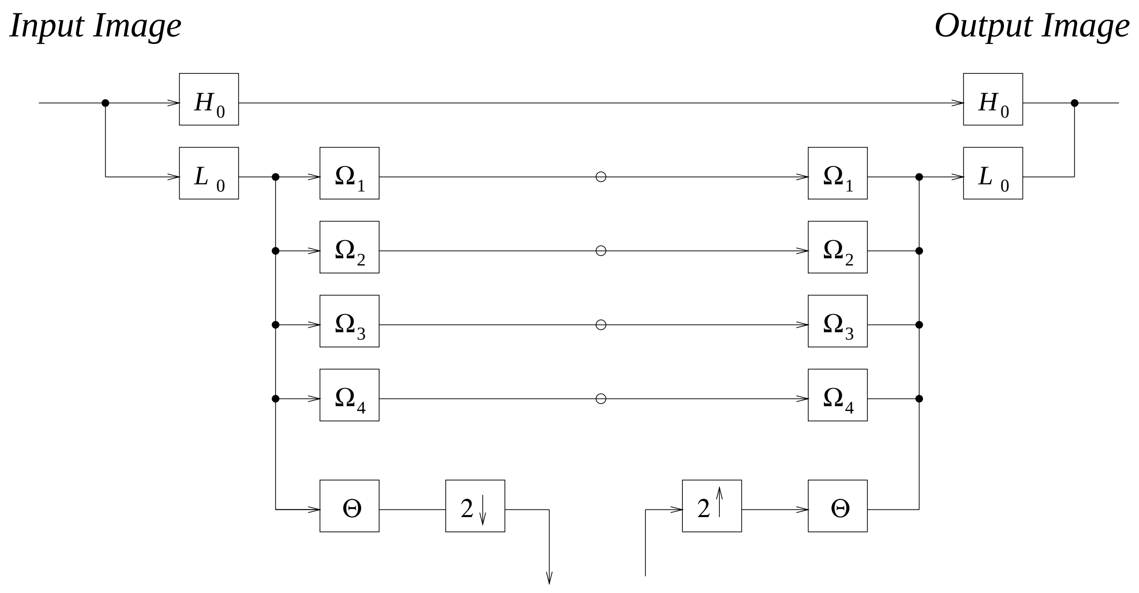


Figure 3: System diagram for the first level of the steerable pyramid. H_0 is a high pass filter and L_0 is a low pass filter. The Ω_i are oriented bandpass filters and Θ is the lowpass filter.

4 Statistical Learning Algorithm

In this section we describe a general algorithm for sampling the most likely mutual source of stationary n-dimensional signals. Section 5 describes the specialization of this algorithm to the tasks of synthesizing and mixing two-dimensional textures. Section 6 describes the specifications of synthesizing sound textures and movie textures using this algorithm.

The outline of our approach is as follows: given k n-dimensional signals as input we construct a 2^n-ary tree representing the wavelet-based multi-resolution analysis (MRA) of each signal. From the point of view of our synthesis algorithm, each signal is now encoded as a collection of paths from the root of the tree towards the leaves. It is assumed that all the paths in a particular tree are realizations of the same stochastic process. The task of the algorithm is to generate a tree whose paths are typical sequences generated by the most likely mutual source of the input trees. From the resulting tree, a new n-dimensional signal is reconstructed by applying a process inverse to the MRA.

Tree merging. Given k source signals represented by their MRA trees $T_1, \ldots, T_k$ with corresponding priors (weights) $\lambda_1, \ldots, \lambda_k$, such that $\sum \lambda_i = 1$, our algorithm generates a new tree by merging together paths present in the source trees. The algorithm is described in pseudocode in Figure 4. The generation of the new tree proceeds in breadth-first order (i.e., level by level). First, we randomly select one of the input trees T_i (according to the given priors). The root value of T_i along with the values of its children are copied into T. Now, let us assume that we have already generated the first i levels of the tree. In order to generate the $(i+1)$-st level we need to assign values to 2^n children nodes of each node in level i. Let x_i be a value of such a node, and denote by $x_{i-1}, x_{i-2}, \ldots, x_1$ the values of that node's ancestors along the path towards the root of the tree. The algorithm searches the i-th level in each of the source trees for nodes y_i with the maximal length ϵ-similar path suffixes $y_i, y_{i-1}, \ldots, y_j$, where ϵ is a user-specified threshold and $i \leq j \leq 1$. Two paths are considered ϵ-similar when the differences between their corresponding values are below a certain threshold. This computation occurs in the routine **CandidateSet** in Figure 4. One of these candidate nodes is then chosen and the values of its children are assigned to the children of node x_i. In this way a complete new tree is formed.

Improvements. The tree merging algorithm described above requires the examination of $k2^{ni}$ paths in order to find the maximal ϵ-similar paths, for each of the 2^{ni} nodes x_i in level i. However, most of the computation can be avoided by inheriting the candidate sets from parents to their children in the tree. Thus, while searching for maximal ϵ-similar paths of node x_i the algorithm only examines the children of the nodes in the candidate sets that were found for x_{i-1} while constructing the previous level. This improvement is especially important in the case of texture movies as described in details in section 6.

```
Input:   Source trees $T_1, \ldots, T_k$, priors $\lambda_1, \ldots, \lambda_k$, threshold $\epsilon$
Output:  A tree $T$ generated by a mutual source of $T_1, \ldots, T_k$

Initialization:
    Randomly choose $T_i$ according to the priors $\lambda_1, \ldots, \lambda_k$
    Root$(T)$ := Root$(T_i)$
    Child$_j(T)$ := Child$_j(T_i)$ for $j = 1, \ldots, 2^n$

Breadth-First Construction:
    for $i = 1$ to $d - 1$ (where $d$ is the depth of the source trees):
        foreach node $x_i$ on level $i$ of $T$
            foreach $j = 1$ to $k$
                $C_j$ := CandidateSet$(T_j, i, x_i, \epsilon)$
                Randomly choose a node $y_{ij}$ from set $C_j$
            endfor
            foreach $j = 1$ to $k$
                $\Lambda_j := \frac{\lambda_j |C_j|}{\sum_{\ell=1}^{k} \lambda_\ell |C_\ell|}$ ($|C_j|$ is the size of the set $C_j$)
            endfor
            Choose $j$ according to the distribution $\Lambda = \{\Lambda_j\}$
            Copy the values of the children of $y_{ij}$ to those of $x_i$
        endfor
    endfor

procedure CandidateSet$(T_j, i, x_i, \epsilon)$
    Let $x_1, x_2, \cdots, x_{i-1}$ be the ancestors of $x_i$
    foreach node $y_i$ on level $i$ of $T_j$
        Let $y_1, y_2, \cdots, y_{i-1}$ be the ancestors of $y_i$
        $L[y_i] := 0$, sum := 0
        for $\ell = i$ to 1
            sum $+= (x_\ell - y_\ell)^2$
            if $\frac{\text{sum}}{i-\ell+1} < \epsilon$ then $L[y_i]$++ else break
        endfor
    endfor
    $M := \max_{y_i} L[y_i]$
    return the set of all nodes $y_i$ such that $L[y_i] == M$
```

Figure 4: The n-dimensional tree-merging algorithm

4.1 Synthesizing Textures of Arbitrary Dimensions

As defined in Section 2, textural signals can be described as signals that exhibit the following property. Using any window of size larger than some critical size, the “information content” exhibited in the window is invariant to the window’s position within the given sample. Since all the textural information is present at samples that are bigger than the critical size, we should be able to synthesize from such samples textures of arbitrary dimensions.

The algorithm described above constructs an output tree T of the same size as the input trees $T_1, \dots, T_k$. It is easy to construct a larger (deeper) output tree as follows. Let d be the depth of the input trees. In order to construct an output tree of depth $d + 1$ we first use our merging algorithm 2^n times to compute 2^n new trees. The roots of those trees are then fed as a low resolution version of the new signal to the MRA construction routine, which uses those values to construct the new root and the first level of the new MRA hierarchy.

In this manner it is possible to generate sound textures of arbitrary length from a short input sample of the sound. The same is true for our 2D texture synthesis algorithm. From a small input 2D texture we can synthesize much bigger texture (as will be demonstrated in Section 5). However, in the case of 3D TVTs, this approach often results in a noticeable temporal discontinuity.

4.2 Threshold Selection

The threshold ϵ is used in our algorithm as a measure of similarity. Recall that in the original suffix learning algorithm [17], the linear sequences contain discrete values. In order to extend the algorithm to sequences of continuous values we use the following similarity criterion. Two paths from nodes x and y to the root of the MRA tree are considered similar if the difference between their values (at each corresponding node along the suffix of length m of the path) are below a certain threshold. If two paths are similar, we can continue one with values from the other, while still preserving the fact that they emerged from the same stochastic source as shown in [17]. In our implementation of this algorithm we use level-dependent similarity criteria for tree paths. Specifically, lower resolution levels of the tree have looser similarity criteria than higher resolution levels, and therefore a larger threshold is used at lower levels. This adaptive measure was chosen because the human visual system is more sensitive to high frequency information.

The selection of the threshold has a big impact on the outcome of the algorithm. Selecting a larger threshold causes the outcome to differ more strongly from the input (the actual difference depends on the type of the input sample, as well as on the value of the threshold). On the other hand, a small threshold can cause the outcome to be a copy of the input. Thus, by leaving the threshold selection to the user, the user is supplied with a powerful tool to achieve the desired outcome. Usually, the thresholds used for synthesizing structured 2D textures are lower than the ones used for synthesizing unstructured textures. This is due to the fact that in the resulting texture we would typically like to retain the large features of the structured texture. Allowing for too large a threshold in such a texture can “break” such features by incorporating random values in wrong places. In the case of texture movies however, selecting the threshold can prove to be a difficult task, since it is harder for the user to assess the scale of the structure present in the temporal dimension of the sequence. We address this problem in Section 6, and show how to automatically choose the threshold in this case.

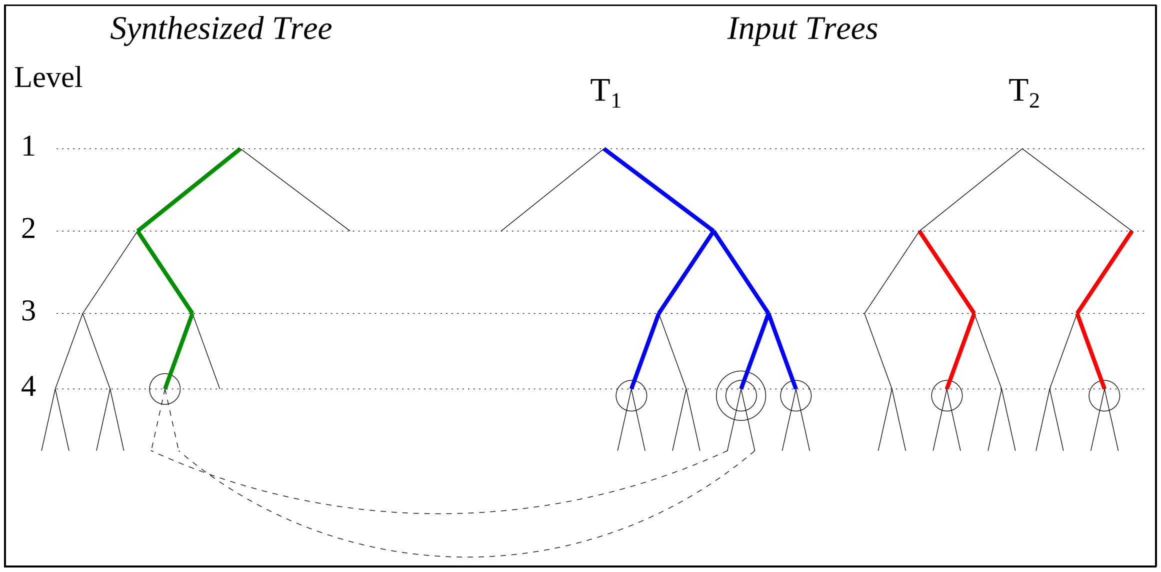


Figure 5: Learning one node in a binary tree

4.3 Example

In Figure 5 we demonstrate our algorithm using an MRA which is a binary tree (as the one we use for the sound texture synthesis in Section 6). We have constructed already the 4 top levels of the synthesized tree (shown on the left). We are now constructing its 5-th level by continuing each node of the fourth level. As can be seen we are doing so from left to right, so some of the nodes in level 5 of the synthesised tree have already been continued. Assume we are now looking to continue the node that is circled on the synthesized tree. We will denote this node by x. On the right are two input trees, that represent the MRA’s of the two input samples. The nodes in these trees that belong to the CandidateSet for x are circled. The maximal similar suffix on those nodes is colored and thickened (most of the nodes that are not in the CandidateSet are omitted from the figure due to lack of space). Note that the length of the maximal similar suffix path on T_1 is 4 (we are counting the nodes, not the edges), while the length of the maximal similar suffix path on T_2 is only 3. Nonetheless, both trees have candidates to continue x. We now choose one of this trees according to the distribution $\Lambda = \{\Lambda_j\}$. Now we choose uniformly one of the candidates in the chosen tree (the double circled node on T_1), and use it to continue x by coping the values of it's children to the values of the children of x. As stated in the algorithm we do this to every node on level 4, and in this way we construct level 5 of the synthesized tree. After d levels, we construct a tree that is in the same depth as the input one. This tree is transformed back to the input sample representation, yielding a new texture.

5 2D Texture Synthesis and Mixing

In this section we describe the implementation of our statistical learning algorithm to the task of synthesizing 2D textures. We also show how one can use this algorithm to synthesize new textures from several different ones, a process that we refer to as “texture mixing”. As mentioned previously, the implementation of our algorithm to the synthesizing of 2D textures from a single input texture appears similar to that of De Bonet [10, 9, 11]. A detailed comparison between the two algorithm is given in this section.

5.1 Texture Synthesis

As explained in the previous section, our statistical learning algorithm begins by constructing an MRA representation for each input sample of the signal. The MRA representation we use to analyze 2D textures is the steerable pyramid [27] described in Section 3.3 with four subband filter orientations (0, 45, 90, and 135 degrees). Color is handled by treating each of the three (red, green, and blue) channels separately. Thus, each node in the MRA tree contains a vector of length 12 (4 filter responses for each of 3 color channels). To obtain the k input trees for our tree merging algorithm, we select k large regions in the input texture (overlapping and slightly shifted with respect to each other). A steerable pyramid is then constructed for each region, yielding k MRA trees. The trees are assigned equal priors of $1/k$. In practice, we found that two or three regions are sufficient for satisfactory results. Our learning algorithm uses these trees to generate a new random MRA tree, which is then transformed back into a 2D image.

Three different examples of textures synthesized by our algorithm are shown in Figure 6. Each row shows a pair of images: the left image is the original texture from which the source trees were generated, and the right image is a synthetic texture, larger than the source by a factor of two in each dimension. Additional examples can be found at:
http://www.cs.huji.ac.il/˜zivbj/textures/textures.html

5.2 Texture Mixing

Texture mixing is a process of generating a new texture that contains features present in several different input textures. Our statistical learning algorithm can be used to mix textures in new creative and interesting ways. Instead of feeding the algorithm with several samples of the same texture, we provide it with samples of several different textures. Since our algorithm produces a new MRA tree by sampling the mutual source of *all* the input trees, the resulting texture exhibits features from all the input textures. Note that this method is

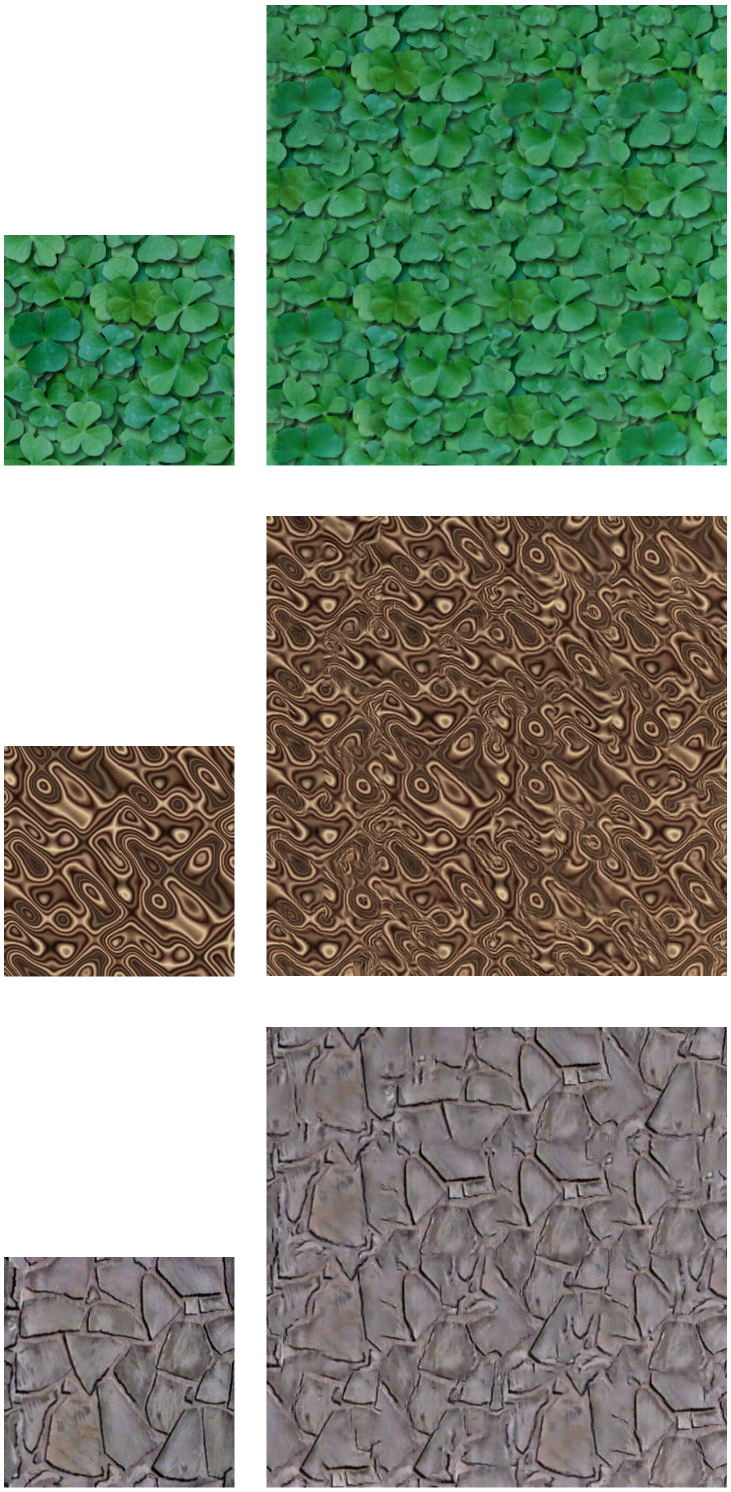


Figure 6: Texture synthesis examples. The synthesized textures are four times larger than the input ones. As described in Section 4, these larger textures were generated by applying our algorithm four times on the same input. Because our algorithm generates a different texture each time, the enlarged texture does not appear tiled.

different from simply averaging the transform coefficients, since each value comes from exactly one input tree. This allows the algorithm to produce a texture that has features from all input textures, while the merging still looks natural (due to the constraints imposed by the statistical learning algorithm). Figure 7 demonstrates the differences between simple texture blending (7a), blending of MRA coefficients *a la* Burt and Adelson [6] (7b), and our technique (7c). Two additional texture mixing examples are shown in Figure 8.

The following problem may arise during texture mixing. When the input textures are very different from each other, the algorithm tends to "lock on" to one of the input trees very early. As a result, starting from a high level node in the generated MRA its entire subtree comes from only one input texture. The result is a large non-mixed area in the output image. A possible solution to this problem is to increase the threshold, relaxing the similarity constraints on the path continuations. However, this solution often results in a blurry outcome.

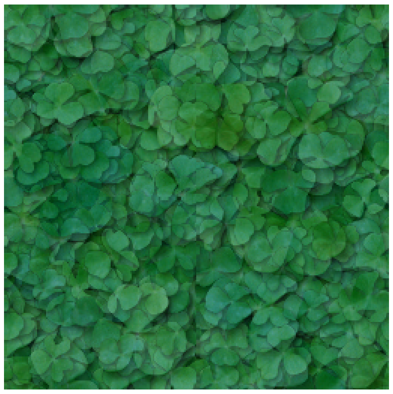
In order to solve the locking problem we try to allow at least one candidate from each input tree to participate when selecting the continuation of a path in the synthesized tree high levels. In order to still be able to preserve strong large features we compute and store in each node of the input trees the cumulative sum of absolute values along the path from the root to that node. This value tends to be large in areas where a strong edge feature is found. When choosing among different candidates, we increase the probability of a candidate according to the magnitude of its cumulative sum.

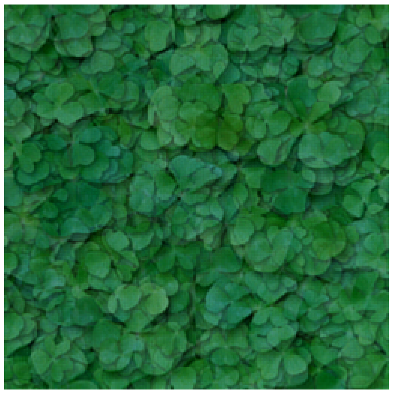
More specifically, we modify our algorithm as follows. When looking for candidates to continue a node x we make sure that there is a non-empty candidate set for each of the input trees (the threshold is increased until the candidate set becomes non-empty). A single candidate is uniformly chosen from each candidate set, as before. Now, instead of choosing among these candidates based on the distribution $\Lambda = \{\Lambda_j\}$, we choose the candidate with the highest cumulative sum.

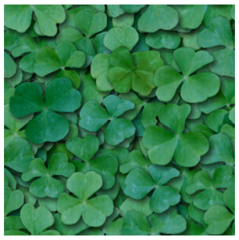
Note that this modified method prefers learning from input tree nodes supporting regions in the texture that exhibit a strong global structure. This is so, because in such regions the steerable filter has a strong response in *all* levels. In smoother regions any of the input textures can be learned. For example, in a brick wall the edges between the bricks will be learned from the brick texture, while the texture inside the bricks might come from other less structured textures.

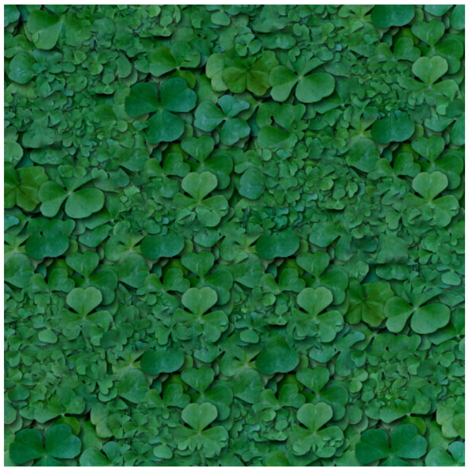
5.3 Comparison With De Bonet's Algorithm

As mentioned earlier the application of our algorithm to the task of 2D texture synthesis is similar to De Bonet's method [10]. However, there are in fact several important differences


(a) simple blend


(b) multi-resolution blend


texture A


(c) mixed texture

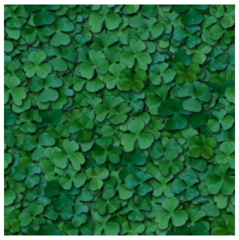
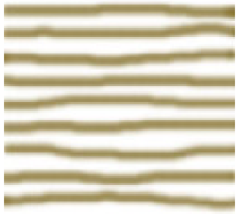
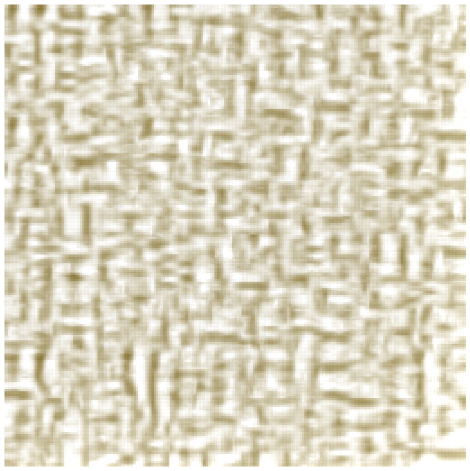
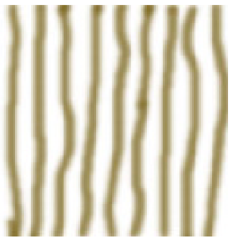
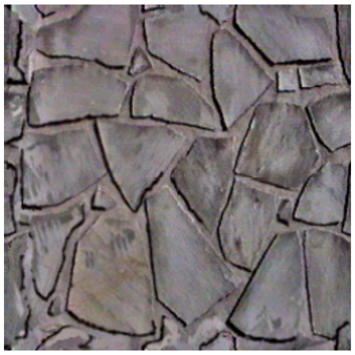

texture B

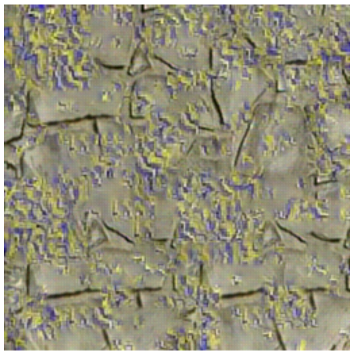
Figure 7: Several different ways of mixing texture A (bottom left) with texture B (bottom right). The top row shows two different blends of A and B: image (a) was obtained by a simple blend of the two textures; image (b) was obtained by blending the coefficients of the multi-resolution transforms of A and B. Compare these blends with the mixed texture (c) produced by our algorithm. In (c) one can discern individual features from both input textures, yet the merging appears natural.


texture A


mixed texture


texture B


texture A


mixed texture

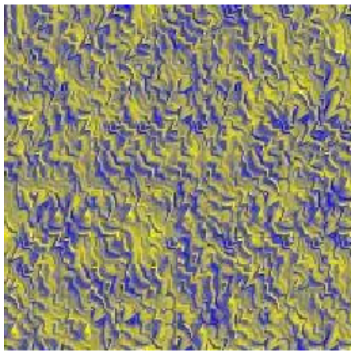

texture B

Figure 8: More texture mixing examples. Note that in the mixed textures one can discern individual features from both input textures, yet the merging appears natural, as opposed to blending or averaging of features.

between the two approaches. After explicitly stating our algorithm, we can now explain those differences in detail.

The most important distinguishing feature of our algorithm is that our tree merging routine takes several texture samples as input and constructs a texture that could have been produced by a mutual source of the input samples. In contrast, De Bonet's algorithm operates on a single texture sample. Using several input samples makes our algorithm more robust and less sensitive to the position of specific features in the input texture. Another practical implication of this difference is that our approach enables us not only to generate textures similar to the input one, but also to synthesize mixtures of different textures, as illustrated in Section 5.2. It should be noted that De Bonet and Viola briefly mention the possibility of extending their approach to multiple input examples [12].

A second difference is that when our algorithm generates level i tree nodes, we are actually looking at nodes in level $i-1$. For each such node x we are looking for nodes in the analysis pyramids (in the same level $i-1$) that have paths similar to x. Once we choose a candidate node x' from this set, we copy the values of *all* the children nodes of x' to the children nodes of x. Since, according to our algorithm, x and x' have the same stochastic source, we would like to imitate this source when generating level i values, thus generating all the children values together. In contrast, when De Bonet's algorithm generates level i nodes, each node in that level is generated separately and independently of its siblings. This can result in more discontinuities in the synthesized texture.

A third difference between our approach and De Bonet's is in the selection of the candidates from which a continuation of a node x is chosen. In both methods, candidates are chosen based on similarity between paths in the tree. De Bonet's method always considers the entire path from the root of the tree to the candidate node. In contrast, our method looks for the ϵ-similar paths of maximal length, including those that do not reach all the way up to the root. Thus, it is possible to choose nodes with paths that do not have similar values in the top levels, but only in the lower levels. As a result, we consider more candidates, allowing for a more varied texture.

See Figures 9 and 10 for a graphic explanation of the differences described above. In practice, the results produced by our synthesis algorithms are of comparable quality to those shown on De Bonet's web pages at:
http://www.ai.mit.edu/ jsd/Research/Synthesis/SPSynth

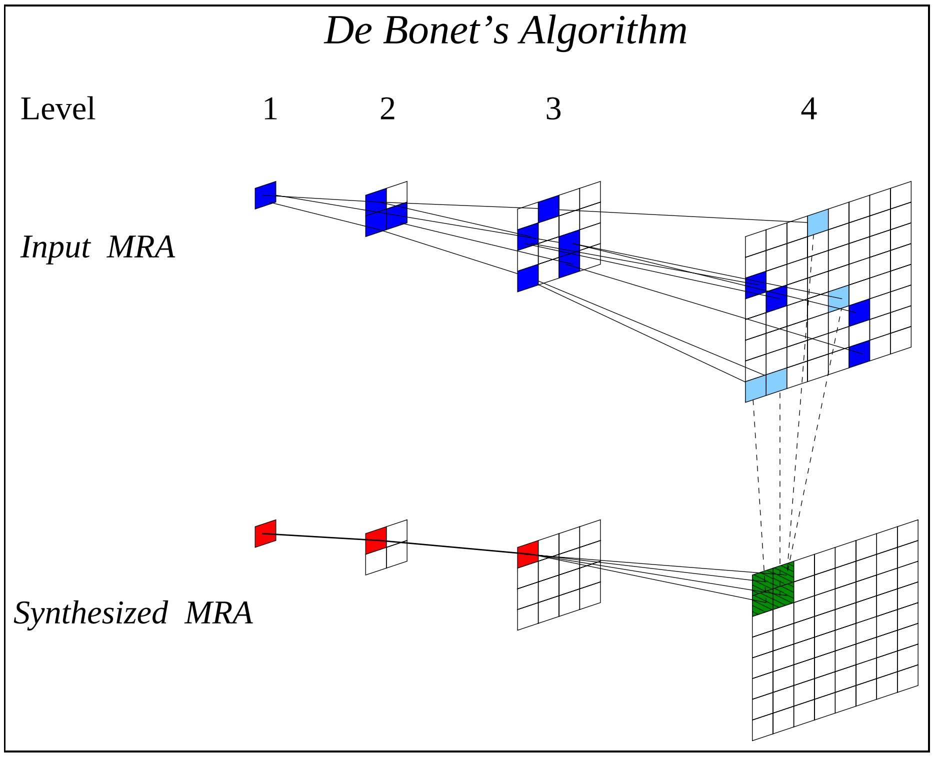


Figure 9: Learning the first 4 nodes of level 4 in the synthesized texture using De Bonet's algorithm. The red nodes are the path from each of the new synthesized nodes to the root. The green nodes are the new synthesized nodes. The blue nodes are the nodes in path's that are similar to the red path in the input textures. The light blue nodes are the nodes from which we copy the values to the new nodes in the synthesized tree. Note the three differences. (1) Only one input texture. (2) The chosen nodes to continue the red path are taken from different places in level 4 of the input textures MRA and do not share the same father. (3) All candidate path's must be similar up to the root. See the next figure for comparison with our algorithm.

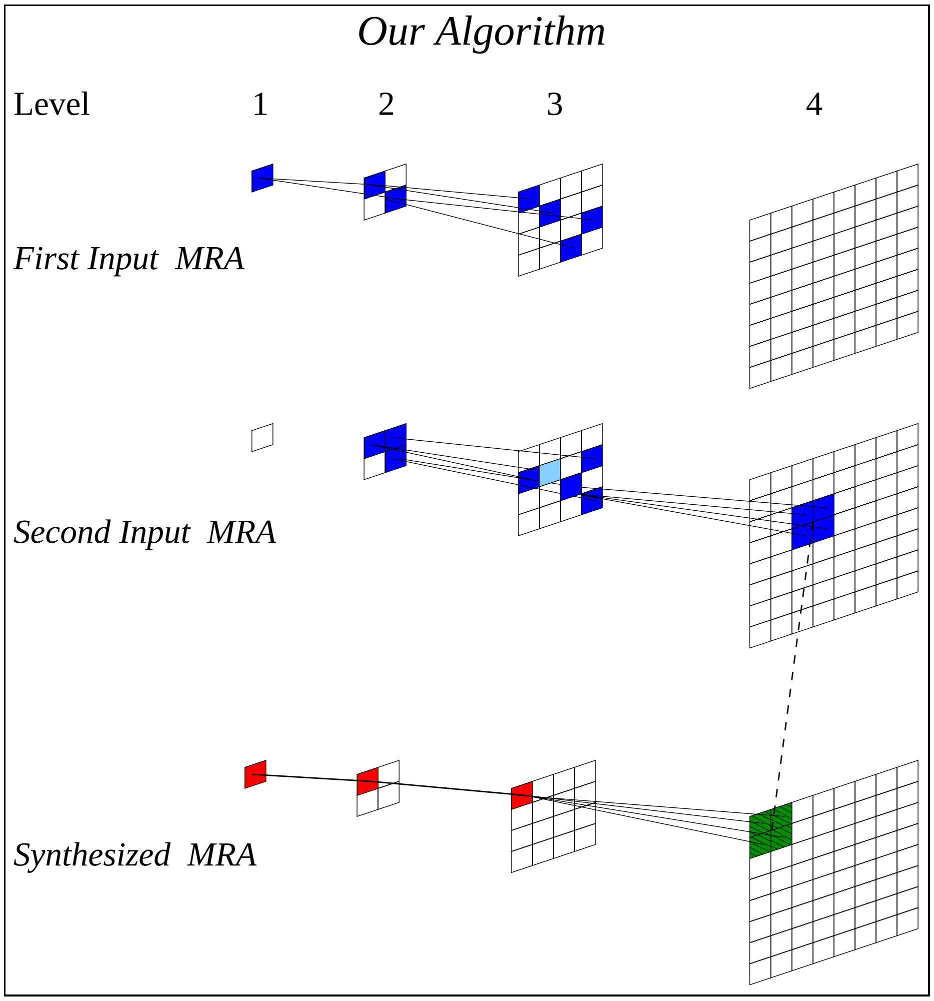


Figure 10: Learning the first 4 nodes of level 4 in the synthesized texture using our algorithm. The red nodes are the path from each of the new synthesized nodes to the root. The green nodes are the new synthesized nodes. The blue nodes are the nodes in path's that are similar to the red path in the input textures. The light blue node is the node in level 3 from which we copy all its sons values to the new synthesized nodes. Note the three differences. (1) More than one input texture. (2) The chosen nodes to continue the red path must share the same father. (3) Arbitrary length of similar path's are allowed, in the first input the length of the similar path is 3 while in the second it is only 2.

6 Synthesis of Time-Varying Textures

This section mainly deals with the application of our algorithm to the task of synthesizing texture movies. A naive approach to generate such movies is to use a our 2D texture synthesis algorithm in order to independently synthesize each frame from the corresponding frame in the input sequence. However, this method fails to capture the temporal continuity and features of the input segment, as demonstrated in the video accompanying this work. In contrast, the approach we will present in this section synthesizes all three dimensions of the TVT simultaneously.

6.1 Differences Between 2D Textures and TVTs

In this section we show how to extend methods for 2D texture synthesis to the case of texture movies. This extension is by no means straightforward, as several important issues must be dealt with.

The most significant difference between the two problems stems from the fundamental difference between the temporal dimension of TVTs and the two spatial dimensions. In a 2D texture, one cannot define a single natural ordering between different pixels in the image: a human observer looks at all the entire image, rather than scanning the image from left to right, or from top to bottom. Thus, the x and y dimensions of the image are treated in the same way by 2D texture synthesis methods. In contrast, there is a clear and natural ordering of events in a texture movie, and a human observer watches the sequence in that order (from the first frame to the last). This indicates that the temporal dimension should be analyzed differently from the spatial ones.

Another practical difficulty with TVT synthesis stems from the higher dimensionality of the texture. A naive extension of the 2D filters used to analyze texture images into 3D drastically increases computation time. A naive extension of the 2D synthesis methods to 3D results in prohibitive synthesis times. Therefore, we have introduced various modifications both in the analysis of the input, and in the synthesis of the output TVT.

Another problem is the selection of the threshold for the temporal dimension. The threshold has a big impact on the resulting TVT, and unlike the 2D case cannot be determined by simply viewing the TVT. In this section we present a method that automatically selects the threshold, based on the characteristic of the input TVT, as will be explained later.

Finally, the methods that deal with 2D texture synthesis usually operate on a $n \times n$ image. However, in the texture movies case the input movie usually has dimensions $n \times n \times r$ where $r < n$. This must be properly accounted for in the analysis and synthesis stages.

6.2 MRA Construction for Texture Movies

As explained in Section 4, the first step in our approach is the construction of a multi-resolution analysis (MRA) of the input signal. In the case of texture movies, the input signal is three-dimensional and hence we construct the MRA by applying a 3D wavelet transform to the signal. The goal of this transform is to capture both spatial and temporal characteristic features of the signal at multiple scales. Since the steerable pyramid transform was found very well-suited for the analysis of 2D textures, our first inclination was to use a 3D variant of the steerable pyramid. However, steerable filters are non-separable and have a wide support (the 2D steerable sub-band filters we used were 9×9). Repeated convolution of a 3D signal with multiple $9 \times 9 \times 9$ 3D filters is quite expensive: it requires $2 \cdot 9^3 = 1458$ floating point operations for each pixel in each frame. Also, designing a set of properly constrained filters for the construction of a steerable pyramid is not a trivial task even in 2D [21]. Finally, in the case of TVTs the signal has different characteristics along the temporal dimension from those it exhibits in the spatial dimensions. While in the temporal dimension there is a clear natural ordering of the frames, there is no prominent natural ordering of the pixels in a single frame. Thus, it does not necessarily make sense to use filters that are symmetric in all three dimensions.

We have also experimented with separable 3D wavelet transforms defined as a cartesian product of three 1D transforms, but they failed to adequately capture the spatial constraints between features in the TVT. The resulting sequences often exhibited strong discontinuities and blocky appearance.

Because of the above considerations, we decided to use a 3D transform defined by a cartesian product between the 2D steerable transform (applied to the spatial dimensions) and an orthonormal 1D wavelet transform (applied to the temporal dimension). Thus, our transform is semi-separable: it is non-separable in 2D, but the time dimension is separable from the other two.

6.2.1 Building the Pyramid

Assume for now that the input signal S is given as a cubic 3D array of size $n \times n \times n$ (we shall lift this restriction in Section 6.2.2). One way to think of this 3D array is as a stack of 2D slices, where each slice is a frame in the sequence. An alternative way to think about it is as a bundle of n^2 1D arrays, where each such array is a temporal sequence of values corresponding to a particular location (x, y).

In order to describe the MRA construction procedure we use the notation introduced in Section 3.3. Let Φ and Ψ denote the scaling function and the wavelet analysis filters of the 1D wavelet transform, respectively. Each of these filters is convolved with a sequence

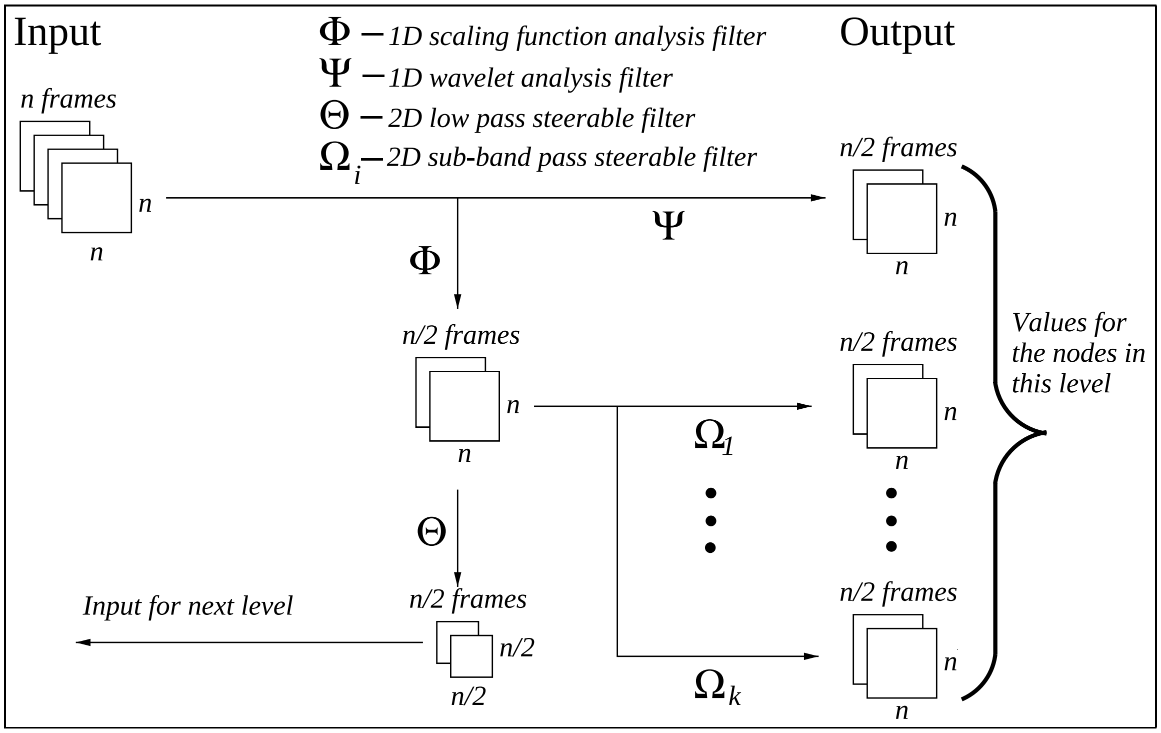

Input
Φ − 1D scaling function analysis filter
Ψ − 1D wavelet analysis filter
Θ − 2D low pass steerable filter
Ω_i − 2D sub-band pass steerable filter
Output
n frames
n
n
Ψ
Φ
n/2 frames
n
n
Values for the nodes in this level
n/2 frames
Ω_1
Θ
Input for next level
n/2 frames
n/2
n/2
Ω_k

Figure 11: Construction of one level of the MRA

of length N to produce $N/2$ coefficients. Let Ω_i denote the i-th sub-band steerable filter, which is convolved with an $N \times N$ image to produce an $N \times N$ response. Let Θ denote the steerable low-pass filter, which is convolved with an $N \times N$ image to produce an $N/2 \times N/2$ low-passed image (the convolution is followed by downsampling by a factor of 2).

The construction of the MRA pyramid proceeds as follows (see Figure 11). Given a 3D TVT we first apply once the analysis filters Φ and Ψ on the n^2 temporal sequences $S(x, y, *)$. The asterisk symbol $*$ is used here to denote the full range of values between 1 and n. Thus, $S(*, *, t)$ stands for the entire t-th frame (time slice) of the signal, while $S(x, y, *)$ denotes the vector of values of the (x, y) pixel in all of the different time frames. Each temporal sequence of length n is thus decomposed to $n/2$ scaling coefficients S^{Φ} and $n/2$ detail coefficients S^{Ψ}. Viewing the $n^2 \times n/2$ scaling coefficients as a stack of $n/2$ slices, the 2D steerable transform is now applied $n/2$ times, once on each slice. Each steerable transform results in k sub-band responses S^{Ω_i} and in a single downsampled low pass response S^{Θ}. All of the detail coefficients S^{Ψ} and the sub-band response values S^{Ω_i} are stored as the values of the nodes at the bottom level of the pyramid. The same procedure is then repeated again to the downsampled low pass responses in order to compute the next level of the pyramid. The pseudocode for this procedure is given in Figure 12.

After constructing the ℓ-th level of the MRA we are left with $n \times n \times n/2$ nodes for this level and a downsampled low pass version of the signal $S^{\Theta}(*, *, *)$, which is then fed again to the same procedure to compute level $\ell - 1$, and so forth. Since S^{Θ} is downsampled by a factor of two in each of the three dimensions, each node in level $\ell - 1$ is considered a parent for eight nodes in level ℓ. Eventually, we construct level $\ell = 1$, where we have four nodes and a single value representing the low pass average of the entire sequence. This value is stored at the root (level zero) of the tree. Thus, we obtain a tree whose root has four children, but otherwise it has a branching factor of eight. Each internal node of the tree contains $3(k + 1)$ values: k subband responses and one detail coefficient for each of the three color channels.

```
Input: A 3D signal $S(*,*,*)$ of size $n \times n \times n$

Output: a) Level $\ell$ of the MRA ($\ell = \log n$)
        b) A low-passed 3D signal $S^{\Theta}(*,*,*)$ of size $\frac{n}{2} \times \frac{n}{2} \times \frac{n}{2}$

Stage 1: apply 1D wavelet transforms
  foreach pixel $(x,y)$
      $\left[S^{\Phi}(x,y,1),\dots,S^{\Phi}(x,y,\frac{n}{2})\right] := \Phi\left(S(x,y,*)\right)$
      $\left[S^{\Psi}(x,y,1),\dots,S^{\Psi}(x,y,\frac{n}{2})\right] := \Psi\left(S(x,y,*)\right)$
  endfor

Stage 2: compute 2D steerable transforms
  for $t = 1$ to $n/2$
      foreach sub-band $i$
          $S^{\Omega_i}(*,*,t) := \Omega_i(S^{\Phi}(*,*,t))$
      endfor
      $S^{\Theta}(*,*,t) := \Theta(S^{\Phi}(*,*,t))$
  endfor

Stage 3: assign values to nodes
  for $t = 1$ to $n/2$
      foreach pixel $(x,y)$
          $Level[\ell].Node[x,y,t].value[0] := S^{\Psi}(x,y,t)$
          foreach sub-band $i$
              $Level[\ell].Node[x,y,t].value[i] := S^{\Omega_i}(x,y,t)$
          endfor
      endfor
  endfor
```

Figure 12: Constructing the ℓ-th level of the MRA

In order for us to be able to learn the new TVT based on the MRA representation of the input TVT, it is imperative that the $3(k+1)$ values stored in each tree node are responses corresponding to the same location in the input signal. This is indeed the case in our construction. We associate $S^{\Psi}(x,y,t)$ and $S^{\Omega_i}(x,y,t)$ with the same node in the MRA. Note that $S^{\Psi}(x,y,t)$ represents the responses of the pixels in the original frames $2t-1$ and $2t$ at location x,y to the temporal filter (since we subsample by a factor of two). $S^{\Omega_i}(x,y,t)$ represents the responses of the same pixels to the steerable filter, since it was applied to the scaling coefficients corresponding to the same pixels.

Since both the 1D orthonormal wavelet transform and the steerable transform are invertible, so is our 3D transform. More precisely, given a TVT of dimensions $n/2 \times n/2 \times n/2$ that was reconstructed at level $\ell-1$ we reconstruct level ℓ in the following way: First we apply the inverse steerable transform on each of the $n/2$ slices of size $n/2 \times n/2$ using the values of the steerable subband responses that are stored in the nodes of level ℓ. This results in $n/2$ slices each of size $n \times n$. We now apply the inverse temporal filter using the values of the highpass temporal filter that are stored in the nodes of this level. This results in an $n \times n \times n$ TVT. We repeat this process until we obtain a TVT of the same size as the input one.

6.2.2 Handling Non-Cubic TVTs

The MRA construction algorithm described above assumes that the input signal has dimensions $n \times n \times n$, where $n = 2^m$. In practice, the input TVT is typically of dimensions $n \times n \times r = 2^m \times 2^m \times 2^q$, where $q < m$. For example, most of the sequences we experimented with were $256 \times 256 \times 32$.

There are many possible strategies to handle non-cubic TVTs. We have experimented with the two strategies described below.

1. Apply the 3D transform from the previous section q times, until S^{Θ} becomes a 2D signal of dimensions $2^{m-q} \times 2^{m-q}$. The remainder of the pyramid is constructed using only the 2D steerable transform. Thus, the resulting pyramid has branching degree 8 in its q bottom levels, and branching degree 4 in the remaining levels.

2. Apply the 2D steerable filters $m-q$ times to each frame, generating $m-q$ levels of the steerable pyramid for each image. We are now left with a $2^q \times 2^q \times 2^q$ signal, and apply our 3D transform to it. Thus, the resulting pyramid has branching degree 4 in its $m-q$ bottom levels, and branching degree 8 in the remaining levels.

We chose to implement the second strategy. We believe that this strategy produces better results because in the resulting tree the nodes containing both temporal and spatial response values are located closer to the root. Thus, all three dimensions of the sequence are taken into account at the early stages of the learning process, i.e., when the overall structure of the output TVT is being formed. The finer spatial details of each frame are filled in later, without any further temporal constraints.

More specifically, when working with the second strategy using our learning algorithm, the algorithm learns *all* the dimensions at the low levels, since using the second strategy results in a low-pass version of the TVT in its lower q levels. This allows us to shape the overall structure of the TVT in its lower levels. After we learn level q, the temporal structure of the TVT is finished and the shape of each frame is well structured. All we have to do from now on is to fill in the details of each frame. This way, each of the three dimensions of the TVT has influence on the output TVT from the first level. Unlike this, using the first strategy, when learning the $m-q$ lower levels of the TVT, we are only influenced by the two dimensional constraints. It is unclear how this structure relates to the temporal dimension of the TVT. Since we learned the lower levels using only two dimensional constraints, it could be that we reduced the possibility of finding a reasonable temporal extension to this representation. This could result in a TVT with low connection between its frames, and can cause a 'jump' between two successive frames.
Indeed, in experiments we made, we found the second strategy to be much better, and this is the strategy we chose to use.

6.3 Synthesis Algorithm

As mentioned in the beginning of this section, we add methods to our statistical learning algorithm in order to deal with TVTs. We describe in this subsection two methods that allows our algorithm to deal efficiently with TVTs. The first one is the implementation of an automatic threshold selection for the temporal domain. The second is an improvement to our algorithm that allows as to handle TVTs in reasonable time.

6.3.1 Threshold Selection

As described in Section 4 two paths are considered similar by our algorithm when the differences between their values are below a certain threshold. The threshold controls the amount of randomness in the resulting signal, and its similarity to the input. In the extreme, too small a threshold may cause the result to be an exact copy of the input. Thus, the value of the threshold has a large impact on the outcome. Intuitively, the thresholds used for synthesizing a structured texture should be lower than those used for synthesizing unstructured ones. This is due to the fact that in the resulting texture we would like to keep the big features of the structured texture. Allowing a large threshold fails to preserve these features by incorporating random values in wrong places.

We cannot expect the user to select an appropriate threshold for the temporal dimension of 3D TVTs, because it is difficult to assess the size of the temporal features in the sequence simply by observing it. Our technique for choosing a threshold for the temporal dimension is inspired by wavelet compression methods for images [13]. The idea behind wavelet compression is to zero out coefficients with L_1 norm less than some small number a. This decimation of the coefficients results in little perceptual effect on subjective image quality. By the same token, we assume that switching is permitted between coefficients whose values are no more than $2a$ apart. Thus, we let the user specify a percentage p. We then compute the interval $[-a, a]$ containing p percent of the TVTs temporal coefficients. The temporal threshold value is then set to $2a$.

6.3.2 Reducing the Number of Candidates

A naive implementation of the tree synthesis algorithm described above requires the examination of *all* the nodes at level i in the original tree in order to find the maximal ϵ-similar paths for every node v_i on level i in the new tree. Given an $2^m \times 2^m \times 2^q$ input TVT, in the bottom level our algorithm has to check $2^{m-1} \times 2^{m-1} \times 2^q$ nodes in the new tree, so applying the naive algorithm results in a number of checks that is quadratic in this number. Since each node has $3k$ values, and for each such value we check a path of length $m - 1$, this exhaustive search makes the synthesis of high-dimensional signals impractically slow. However, as briefly mentioned in Section 4, much of the search can be avoided by inheriting the candidate sets from parents to their children in the tree. Thus, while searching for maximal ϵ-similar paths of node v_i the algorithm must only examine the children of the nodes in the candidate sets that were found for v_{i-1} while constructing the previous level. The result is a drastic reduction in the number of candidates. The actual number of candidates depends of course on the threshold, but in almost all cases we found that the number is very small (between 4 and 16). In the case of our 3D TVTs we found that this improvement reduced the synthesis time from weeks to just a few minutes.

6.4 Results

We applied the algorithms described in this work to the synthesis of two kinds of TVTs: texture movies and sound textures. The accompanying videotape contains several examples of both kinds. Generally, for each example we first show the synthesized movie clip several times in a loop (sometimes accompanied by a soundtrack). Next, a side-by-side comparison with the original clip is shown. It should be noted that all of the sounds accompanying the synthetic movie segments on the videotape were synthesized by our algorithm.

Waterfall See rows 1 and 2 in Figure 13. The differences between the original and synthesized clips are noticeable both in the static structure of the waterfall, and in the water flow. These differences are particularly apparent in the left hand side and at the top of each frame.

Crowd See rows 3 and 4 in Figure 13. In this example, the videotape also shows a failed attempt to generate a continuous sequence by synthesizing each frame independently using a 2D texture synthesis algorithm, i.e., without temporal constraints.

Volcano See rows 5, 6, and 7 in Figure 13. Here we show two different synthesized clips. A slow-motion side-by-side comparison is also provided on the videotape.

Sound textures We show three examples. In each example we first show the original waveform accompanied by the corresponding sound, and then the synthesized waveform and sound. The sounds are: a drum beat, a baby crying, and cars in a traffic jam. The synthesized drum beat sequence is longer than the input sample. Figure 15 shows a graphic comparison for this example. It can be seen that the two signals are clearly different from each other, but both contain the same essential features.

Clouds See rows 1 and 2 in Figure 14.

Fire See rows 3 and 4 in Figure 14.

Jellyfish See rows 5 and 6 in Figure 14.

6.4.1 Implementation Specifics

The 1D orthonormal wavelet used for the analysis along the temporal dimension is a Daubechies filter of length 10 [8]. For the spatial domain analysis we used the steerable pyramid [27] with four subband filter orientations (0, 45, 90, and 135 degrees). Color is handled by treating the red, green, and blue components separately. Thus, the values at each node of the MRA hierarchy are vectors of length 15 in its high levels (where we analyze the TVT as a cube) and of length 12 in its lower levels. The threshold for the temporal responses was obtained as described in Section 6.3.1 when p was usually between 70 and 80 percent. With our current implementation we generated movies of size $256 \times 256 \times 32$ on a Pentium II 450MHz with 1GB of RAM. Each movie clip takes about 10 minutes to generate.

In the case of sound textures, which are 1D signals, we built the MRA using a 1D wavelet transform. Thus, the MRA that was obtained is a binary tree, where each node stores the detail coefficients of the Daubechies wavelet. We used the same Daubechies filter of length 10. The threshold for the learning algorithm was computed as described in Section 6.3.1, when p was usually between 60 to 70 percent.

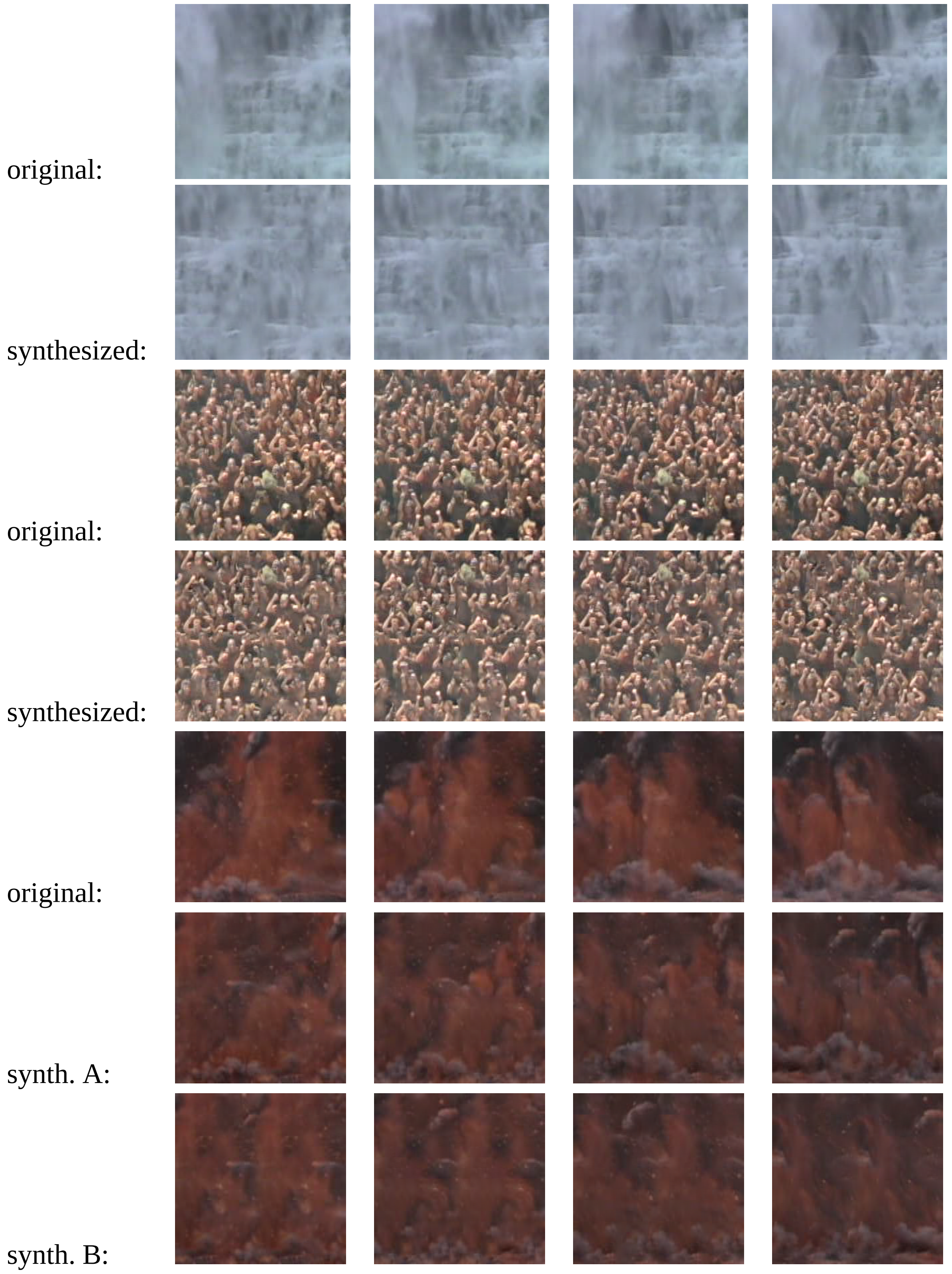


Figure 13: Texture movie synthesis examples. In each of the examples above, the first row shows four frames from the original movie clip (frames 0, 7, 14, and 21), and the following row(s) shows the corresponding frames in the synthesized clip(s). The examples are: waterfall (rows 1–2), crowd (rows 3–4), and volcano (rows 5–7, two different synthesized clips). While the synthesized frames are very similar to the original in their overall appearance, pairwise comparison reveals many differences.

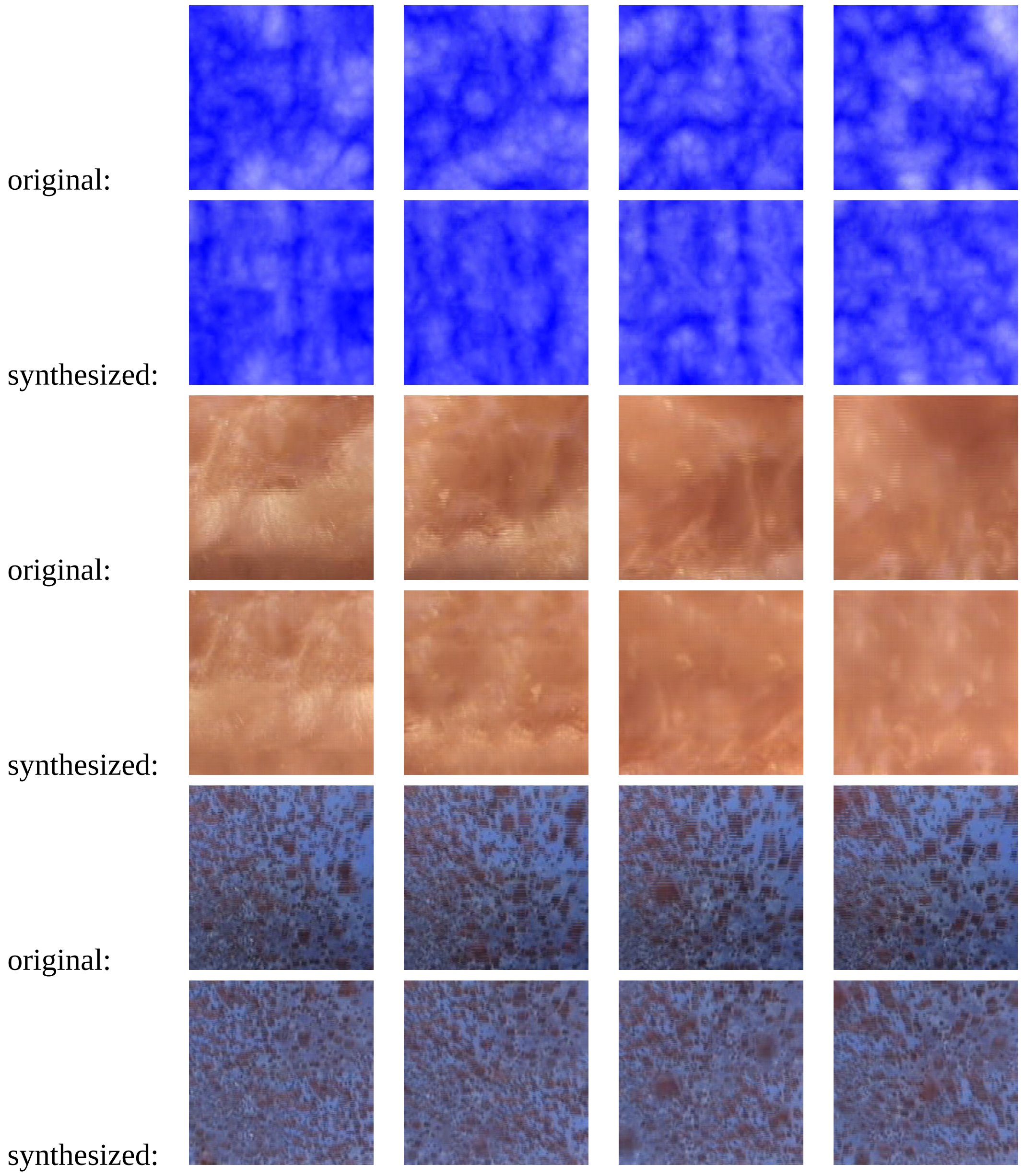


Figure 14: More texture movie synthesis examples: clouds (rows 1–2), fire (rows 3–4), and jellyfish (rows 5–6).

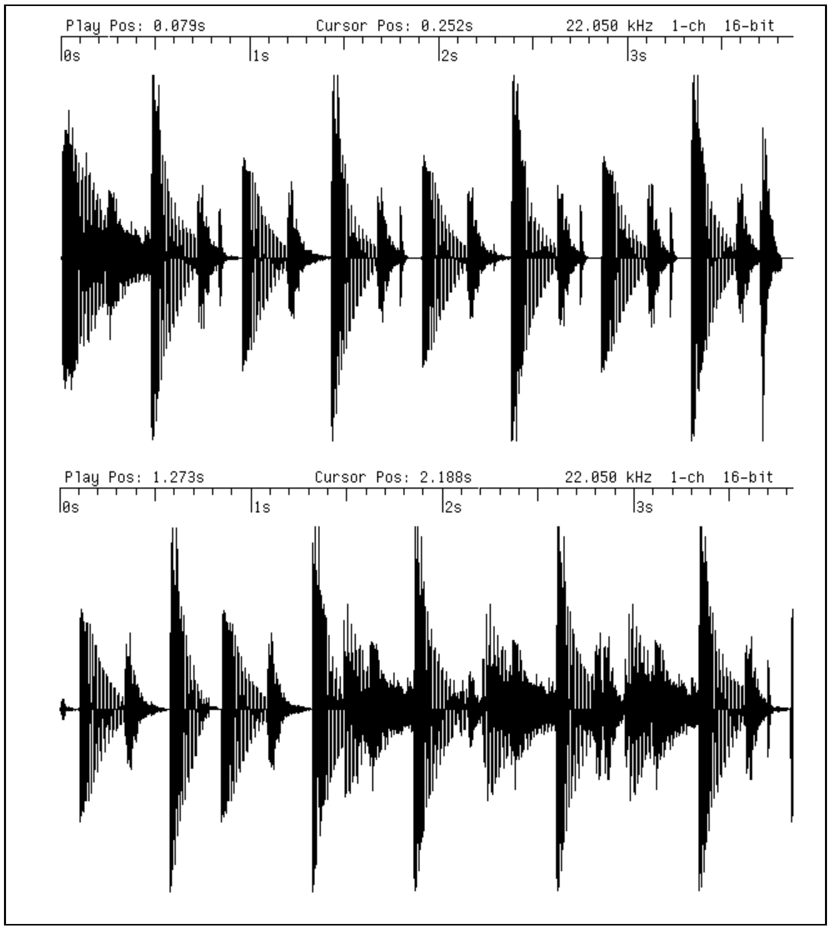


Figure 15: A graphic comparison between an original drum-beat sound waveform (top) and a synthesized one (bottom). Note that the synthesized waveform contains the same features, but without the periodicity of the original signal.

6.4.2 Limitations

Our algorithm stores in main memory both the MRA constructed from the input signal and the MRA that is being generated. The resulting memory requirements are quite substantial. Specifically, on a workstation with 1GB of RAM our implementation currently generates short movies (32 frames at 256×256 resolution, and 128 frames at 128×128). Although longer frame sequences can be generated by creating several short ones and concatenating them while blending their boundary frames, this approach often introduces excessive blur in the blended frames and does not result in a desired "typical" TVT.

As can be noticed in some of the examples on the videotape, occasional spatial and temporal discontinuities can be seen. This results from the tree-based nature of the synthesis algorithm. Neighboring spatio-temporal regions in the movie can sometimes be far apart in the MRA tree structure. In those cases the constraints between such regions are weaker than they should be.

Our approach assumes that the frames are filled with texture in a relatively homogeneous manner; the method does not respond well to large changes in the size of texture features across the frames, which can occur for example due to perspective foreshortening. Large static objects in the field of view also interfere with successful synthesis. The method works best when the camera appears to be stationary.

7 Conclusions and Future Work

7.1 Summary

We have described a method, based on statistical learning and MRA construction, for generating new instances of textures from input samples. While most of the previous work in this area has focused on the synthesis of 2D textures [38, 19, 9], our work is not limited to such textures and it is capable of synthesizing texture movies and sound textures as well. In the case of 2D textures our algorithm can also produce a texture that is a mixture of several input textures. This new texture is statistically similar to all of the input samples and it exhibits a mixture of their features.

Our experiments demonstrate that the algorithm is robust, and works on a large variety of textures. From the same input, we can generate many different textures. For 1D and 2D textures our algorithm is able to produce textures of different sizes from a small input texture. A variation of the same method is used to produce mixed 2D textures that simultaneously capture the appearance of a number of different input textures. The ability to produce such mixes will undoubtedly enhance the creative abilities of artists and graphics designers. Examples of synthesized and mixed 2D textures can be found at
http://www.cs.huji.ac.il/˜zivbj/textures/textures.html

The major contribution of this work is the texture movie synthesis algorithm. To our knowledge, there have not been any previous attempts towards statistical learning of TVTs from input samples. So far, synthesis of dynamic natural phenomena has mostly been possible only via computationally intensive physically based simulations. Such simulations are for the most part expensive and, more importantly, the resulting animations can be difficult to control. In contrast, our statistical learning algorithm for TVTs is extremely general, automatic, and fast alternative, provided that an example of the desired result is available.

There are many possible applications to such a technique in the field of special effects and movie animation. Our algorithm can save a lot of money to movie producers by allowing them to film small scenes with a few people, and then combine many synthesized versions of this scene to create a scene with many people. Since our algorithm produces many different output movies from the same input one, the audience will not get the feeling that this scene is a tiling of many small ones. Another possible application is the filming of one explosion, and producing many different explosions from this one. Those synthesized explosions can be placed at different scenes in the movie without repeating the same explosion twice. Another possible use is the combination of synthesizing scenes using our algorithm, and synthesizing the sound track for those scenes using our 1D synthesis algorithm (as demonstrated in the accompanying videotape). All these possibilities give a powerful tool in the hand of a special effects director, and can help in producing better movies at less time and lower costs.

7.2 Directions for Future Work

The methods described in this work are just the first step towards building a complete system for automatic generation of special effects from examples. There are many ways to further enhance and extend our approach.

Longer movies. At present, our algorithm produces movie clips of the same length as the input clip. Longer clips can be generated by concatenating their MRA trees, but this often results in a temporal discontinuity. Thus, a more drastic change in the algorithm is needed in order to be able to generate arbitrarily long frame sequences. We would like to develop an algorithm capable of adding more frames to a prefix frame sequence that has already been computed, without having to construct the entire MRA tree of the longer sequence.

Full integration of sound and picture. Currently, the synthesis of the movie and its soundtrack are completely independent. We would like to extend our algorithms to take into account constraint between these two modalities, and to synthesize them in a synchronized fashion.

Movie mixing. It should be possible to extend the technique for 2D texture mixing described in Section 5.2 to generation of "movie mixtures".

Classification. Methods for statistical learning of 2D texture images have been successfully applied not only to texture generation, but also to texture recognition and image denoising [11]. These applications are made possible by realizing that the statistical learning of 2D textures implicitly constructs a statistical model describing images of a particular class. Similarly, our approach for TVT generation can be used as a statistical model suitable for describing TVTs. Therefore, it should be possible to apply this statistical model for tasks such as classification and recognition of such movie segments.

References

[1] Wavelets for texture analysis. http://wcc.ruca.ua.ac.be/ visielab/wta/textu.html.

[2] Ziv Bar-Joseph, Shlomo Dubnov, Ran El-Yaniv, Dani Lischinski, and Michael Werman. Statistical learning of granular synthesis parameters with applications for sound texture synthesis. In *International Computer Music Conference (ICMC99)*, 1999.

[3] M. Basseville, A. Benveniste, K.C. Chou, S.A. Golden, R. Nikoukhah, and A.S. Willsky. Modeling and estimation of multiresolution stochastic processes. *IEEE Transactions on Information Theory*, 38(2):766–784, 1992.

[4] M. Basseville, A. Benveniste, and A.S. Willsky. Multiscale autoregressive processes, part II: Lattice structures for whitening and modeling. *IEEE Transactions on Signal Processing*, 40(8):1935–1954, 1992.

[5] G.E.P. Box, G.M. Jenkins, G.C. Reinsel, and G. Jenkins. *Time Series Analysis : Forecasting and Control*. Prentice Hall, 1994.

[6] P.J. Burt and E.H. Adelson. A multiresolution spline with application to image mosaics. *ACM Transactions on Graphics*, 2(4):217–236, October 1983.

[7] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley Series in Telecommunications. John Wiley & Sons, New York, 1991.

[8] Ingrid Daubechies. Orhtonormal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 41(7):909–996, October 1988.

[9] J. S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Computer Graphics*, pages 361–368. ACM SIGGRAPH, 1997.

[10] J. S. De Bonet. Novel statistical multiresolution techniques for image synthesis, discrimination, and recognition. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, May 1997.

[11] J. S. De Bonet and P. Viola. A non-parametric multi-scale statistical model for natural images. *Advances in Neural Information Processing*, 10, 1997.

[12] J. S. De Bonet and P. Viola. Texture recognition using a non-parametric multi-scale statistical model. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, 1998.

[13] Ronald A. DeVore, Björn Jawerth, and Bradley J. Lucier. Image compression through wavelet transform coding. *IEEE Transactions on Information Theory*, 38(2 (Part II)):719–746, 1992.

[14] David Ebert, Wayne Carlson, and Richard Parent. Solid Spaces and Inverse Particle Systems for Controlling the Animation of Gases and Fluids. *The Visual Computer*, 10(4):179–190, 1994.

[15] David Ebert, Kent Musgrave, Darwyn Peachey, Ken Perlin, and Worley. *Texturing and Modeling: A Procedural Approach*. Academic Press, October 1994.

[16] David S. Ebert and Richard E. Parent. Rendering and animation of gaseous phenomena by combining fast volume and scanline A-buffer techniques. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 357–366, August 1990.

[17] Ran El-Yaniv, Shai Fine, and Naftali Tishby. Agnostic classification of Markovian sequences. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[18] Alain Fournier and William T. Reeves. A simple model of ocean waves. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20(4), pages 75–84, August 1986.

[19] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In Robert L. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 229–238. ACM SIGGRAPH, Addison Wesley, August 1995.

[20] Anil K. Jain. *Fundamentals of Digital Image Processing*. Prentice Hall, New Jersey, 1989.

[21] Anestis Karasaridis and Eero Simoncelli. A filter design technique for steerable pyramid image transforms. In *Proc. ICASSP-96*, May 7–10, Atlanta, GA, 1996.

[22] N. Merhav and M. Feder. Universal prediction. *IEEE Transactions on Information Theory*, 44(6):2124–2147, 1998.

[23] Darwyn R. Peachey. Modeling waves and surf. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20(4), pages 65–74, August 1986.

[24] Ken Perlin. An image synthesizer. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 287–296, July 1985.

[25] W. T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, 2:91–108, April 1983.

[26] William T. Reeves and Ricki Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 313–322, July 1985.

[27] Eero P. Simoncelli, William T. Freeman, Edward H. Adelson, and David J. Heeger. Shiftable multi-scale transforms. *IEEE Transactions on Information Theory*, 38(2):587–607, March 1992. Special Issue on Wavelets.

[28] Karl Sims. Particle animation and rendering using data parallel computation. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 405–413, August 1990.

[29] J. R. Smith. *Integrated Spatial and Feature Image Systems: Retrieval, Analysis and Compression*. PhD thesis, Columbia University, 1997.

[30] Jos Stam and Eugene Fiume. Turbulent wind fields for gaseous phenomena. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 369–376, August 1993.

[31] Jos Stam and Eugene Fiume. Depicting fire and other gaseous phenomena using diffusion processes. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 129–136. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[32] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1996.

[33] Greg Turk. Generating textures for arbitrary surfaces using reaction-diffusion. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 289–298, July 1991.

[34] R. Wilson and M. Spann. *Image Segmentation and Uncertainty*. Research Studies Press Ltd, 1988.

[35] Andrew Witkin and Michael Kass. Reaction-diffusion textures. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 299–308, July 1991.

[36] Steven P. Worley. A cellular texture basis function. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 291–294. ACM SIGGRAPH, Addison Wesley, August 1996.

[37] G.W. Wornell and A.V. Oppenheim. Wavelet-based representations for a class of self-similar signals with application to fractal modulation. *IEEE Transactions on Information Theory*, 38(2):785–800, 1992.

[38] S.C. Zhu, Y. Wu, and D. Mumford. Filters random fields and maximum entropy(frame) - towards a unified theory for texture modeling. *Int'l Journal of Computer Vision*, 27(2):107–126, 1998.